Compiling Planning into Scheduling: A Sketch

Tania Bedrax-Weiss* and **James M. Crawford** and **David E. Smith**
Computational Sciences Division
NASA Ames Research Center
Mailstop 269-4
Moffett Field, CA 94035-1000
{tania,jc,de2smith}@email.arc.nasa.gov

Abstract

Although there are many approaches for compiling a planning problem into a static CSP or a scheduling problem, current approaches essentially preserve the structure of the planning problem in the encoding. In this paper we present a fundamentally different encoding that more accurately resembles a scheduling problem. We sketch the approach and argue, based on an example, that it is possible to automate the generation of such an encoding for problems with certain properties and thus produce a compiler of planning into scheduling problems. Furthermore we argue that many NASA problems exhibit these properties and that such a compiler would provide benefits to both theory and practice.

Introduction

Despite significant advances in both the performance and representation capabilities of recent AI planning systems there is still a large gap between the size and complexity of problems that can be handled by planning systems and those that can be handled by scheduling systems. Take for example, the Earth-Observing Satellites domain where the objective is to plan for a collection of observation requests that must be fulfilled by available satellites. In the 3rd Planning Competition (Long & Fox 2003), planners were able to generate plans for 100 observations and a handful of satellites on a simplified version of this domain. Others (Dungan *et al.* 2002; Morris *et al.* 2003) have been able to do somewhat better on the full domain using a search strategy tailored for this domain. In contrast, Globus et al. (2002; 2004) have reported the ability to solve problems involving 6,000 observations (and 100 satellites) in a few seconds by encoding the problem as a scheduling problem using a CSP.

This suggests that, wherever possible, we should reduce planning problems to scheduling problems and solve them that way. Synthesizing the problems by hand, however, may pose many challenges. For one thing, a good scheduling encoding of a problem is difficult to obtain, and may not be very natural or obvious. Furthermore, a scheduling encoding can be quite sensitive to minor changes in the nature of the problem, whereas the planning encoding might be more natural, and easier to adapt to changing domain or problem characteristics.

What we would like then, is to be able to recognize when all or part of a planning problem can be reduced to a scheduling problem, and automatically synthesize a static CSP representation of that scheduling problem. In this paper, we describe an approach for automatically compiling a planning problem into a static CSP under the following conditions: (1) the problem must have many weakly interacting goals or sub-goals; (2) each goal can be achieved in reasonably small number of ways; (3) positive interactions among sub-goals must be rare; (4) goals must be non-cyclic; that is, a subgoal tree for a goal must not contain the goal itself. One problem domain that satisfies these conditions is the problem of scheduling Earth-Observing Satellites (EOS).

The static CSP formulation of the EOS domain has: 1. a predicate that represents a candidate plan for satisfying a sub-goal independently of other subgoals; 2. a disjunctive constraint over these predicates representing the choice of candidate plans for a sub-goal; and 3. mutual exclusion constraints among the predicates, indicating which combinations of plans are incompatible. Thus, a solution to the problem is found when a candidate plan is chosen for each subgoal such that the mutual exclusion constraints are all satisfied.

This formulation of the problem resembles a scheduling problem in that the candidate plans are are to be scheduled according to the mutual exclusion constraints. In contrast to other planning encodings there is no variable or constraint specifically denoting causality between actions and fluents. In fact, there are no variables that correspond directly to actions or fluents (except for resources which can be seen as a special case of fluents but are common in scheduling problems). By abstracting away the planning nature of the problem and making scheduling decisions explicit we believe that we can take advantage of scheduling technology to solve this problem directly. Other approaches would have unnecessary variables describing the causality of the planning problem. These extra variables and constraints may make it harder to guide a search engine to focus on what is important.

We generate multiple candidate solutions for each subgoal by building lifted plan graphs. A lifted plan graph is like a plan graph (Kautz & Selman 1998) except initial and

*QSS Group, Inc.

Copyright © 2004, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

goal conditions are lifted and actions remain lifted as they're added to the plan graph. The result is a set of candidate plans and constraints that specify when each plan satisfies the sub-goal. Mutual exclusion constraints are derived from analyzing all possible ways that plans can interfere with each other. We assume a PDDL 2.1 semantics (Fox & Long 2003) for durative actions. Assuming that it is possible to generate all possible ways of satisfying each goal and that it is possible to identify all possible ways in which each sub-plan can interact with another, then this approach can be used to generate an optimal solution in terms of plan length (or in the case of EOS, in terms of slew time).

Earth-Observing Satellites

Consider the problem of scheduling Earth-Observing Satellites (EOS). Briefly, the EOS problem is to plan a set of image requests such that the number and quality of satisfied requests is maximized and the slewing time is minimized (slew time is the time spent panning the camera from the position required for one image to the position for the next image). Each image is stored onboard the satellite in the Solid State Recorder (SSR) until it is downlinked to the ground.

The EOS problem can be formulated as a planning problem in a straightforward way. Each image requested by the scientists is a goal (more precisely, having the image data in the computer on the ground is a goal). This goal breaks down into sub-goals of slewing the camera to point to the correct spot on the ground, warming up the camera, taking the image, and then downlinking the image to the ground (depending on the level of support from the lower level control architecture these steps may require further sub-goaling but we can ignore this for purposes of this example). Each of these sub-goals is constrained in various ways (e.g., slewing takes some period of time, the camera can only point some number of degrees across or along the satellites track, etc.) and various sub-goals consume resources (e.g., power for the slew, and SSR capacity from the time the image is taken to the time of the downlink).

Existing Planning Encodings

This planning problem can be translated to a SAT problem or more generally a static CSP problem (which can be fully ground into SAT) using a variety of techniques from the literature (Bedrax-Weiss, Jónsson, & Ginsberg 1996; Kautz & Selman 1998; Do & Kambhampati 2000; Mali & Kambhampati 1998). The key insight on which most of these approaches are based is that because both SAT and bounded-length STRIPS-style planning are NP-complete, polynomial reductions between the problems exist. We assume that an optimal solution in terms of number and quality of satisfied requests can be obtained by imposing and subsequently changing the bounds. The slew time, however is determined by the order of the observations so that is computed based on the solution to the SAT formulation.

The details in each approach to planning as SAT differ but the essential idea is to create a variable for the fully ground operation to be performed at each time step. Variables are also created for fluents including SSR levels. Finally, constraints are added to ensure that preconditions, post-conditions, and resource constraints are met. For simplicity we only present fragments of encodings in the literature.

A state based encoding of EOS using explanation-based axioms (Haas 1987) would have constraints of the form:

$$\mathrm{Pointing}(1,t+1) \leftrightarrow \mathrm{Pointing}(1,t) \vee \mathrm{Slew}(2,1,t) \vee \ldots$$
$$\mathrm{ImgTaken}(1,t+1) \leftrightarrow \mathrm{ImgTaken}(1,t) \vee \mathrm{TakeImg}(1,t)$$
$$\mathrm{ConsumeSSR}(1,t) \leftrightarrow \mathrm{TakeImg}(1,t)$$
$$\neg\,\mathrm{TakeImg}(1,t) \vee \neg\,\mathrm{Slew}(1,2,t)$$
$$\ldots$$

where t is instantiated for each allowed value for time. Note that in this encoding, Slew, TakeImg, and ConsumeSSR are instantaneous actions. In fact, we are not aware of any encodings that encode actions with durations. Durations are essential when dealing with resources because it is otherwise awkward and sometimes impossible to precisely model the state of a resource (Bedrax-Weiss, McGann, & Ramakrishnan 2003; Smith 2003). For simplicity we have chosen to model ConsumeSSR as consuming the entire storage rather than modeling the consumption pattern, which would introduce extra predicates and mutual exclusions of the form "only X of the N possible consumption predicates must be true at any one time". Note also that the causality between preconditions and effects of Slew and TakeImg is preserved by the constraints, and that there are variables for each possible effect, precondition and action at each possible time.

There are other encodings that do away with time altogether. A partial order or causal encoding (Kautz, McAllester, & Selman 1996) based on (McAllester & Rosenblitt 1991) encodes the set of actions that can be performed at each step in the plan. A step corresponds to the time in which the action associated to that step is executed. Two different steps can be assigned the same action if the action is to be performed at two different times. Step ordering assertions come from prerequisites and causal links. Prerequisites are given by preconditions that need to be satisfied. Causal links are given by requiring that if a step precedes another step that there be no step between them that either adds or deletes a condition that is protected by the causal link.

A causal encoding of EOS would have constraints of the form:

$$\mathrm{Action}(o1,\mathrm{TakeImg}(1)) \vee \mathrm{Action}(o1,\mathrm{Slew}(1,2)) \vee \ldots$$
$$\neg\,\mathrm{Action}(o1,\mathrm{TakeImg}(1)) \vee \neg\,\mathrm{Action}(o1,\mathrm{Slew}(1,2))$$
$$\mathrm{Action}(o1,\mathrm{TakeImg}(1)) \rightarrow \mathrm{Needs}(o1,\mathrm{Pointing}(1))$$
$$\mathrm{Action}(o2,\mathrm{Slew}(1,2)) \rightarrow \mathrm{Dels}(o2,\mathrm{Pointing}(1))$$
$$\mathrm{Action}(o3,\mathrm{Slew}(3,1)) \rightarrow \mathrm{Adds}(o3,\mathrm{Pointing}(1))$$
$$\mathrm{Action}(o3,\mathrm{Slew}(3,1)) < \mathrm{Action}(o1,\mathrm{TakeImg}(1)) \wedge$$
$$\mathrm{Dels}(o2, \mathrm{Pointing}(1)) \rightarrow o1 < o2 \vee o2 < o3 \ldots$$

where o1, o2, and o3 are step names and the predicate Action(o1,Slew(2,1)) reflects the decision to add an action to slew from 2 to 1 to the plan. Furthermore, it adds constraints for every causal link and ordering constraints between the operators. Both of these translations create SAT encodings where the essential "planning" structure of pre-conditions, post-conditions, and causality are preserved. In fact, other encodings we are aware of such as the HTN encoding (Mali

& Kambhampati 1998) also preserve the planning structure. The HTN encoding contains constraints for task decomposition in addition to the classic constraints for achieving goals.

A Proposed Encoding

A key observation about EOS scheduling is that there are relatively few time "windows" at which images can be taken. This is because the cameras can only slew a relatively small number of degrees, and the number of times the satellite passes a given point within a scheduling period is relatively small. Furthermore, for each image request there is a small set of satellites and instruments that can possibly fulfill it. The nature of the problem is such that the key issue is assigning image acquisition actions to windows of opportunity.

Storage and power consumption for each individual sub-goal depend only on the target that the satellite is currently pointing to and the sub-goal target. Thus, it is possible to associate these costs to each plan for each sub-goal. Furthermore, the cost need not be reflected on the encoding. It can be naturally dealt with in a scheduling/CSP framework.

We choose to work in the CSP setting instead of in the fully ground SAT setting for reasons that we will describe later. The EOS problem can be formulated as a static CSP as follows. For each goal, create a disjunctive list of all possible plans satisfying the goal in isolation of other goals. Then, introduce mutual exclusion relations between pairs of incompatible plans. Ensure observations can only happen within opportunity windows. We assume s, s1, and s2 are variables of type satellite, and x, x1, x2, and y are variables of type observation target.

The constraint that is imposed by the observation windows is:

$$\mathrm{start}(\mathrm{Observed}(s,x)) \geq \mathrm{start}(\mathrm{OppWindow}(s,x)) \wedge$$
$$\mathrm{end}(\mathrm{Observed}(s,x)) \leq \mathrm{end}(\mathrm{OppWindow}(s,x))$$

where Observed(s,x) denotes that satellite s observed target x and start(Observed(s,x)) is a function that represents the start time of Observed(s,x). This constraint is augmented if there is more than a single opportunity window for each observation. Similarly if there's more than one instrument that can perform the observation there will be constraints reflecting the possible choices.

An observation must be satisfied through one of the possible means of achieving it.

$$\mathrm{Observed}(s,x) \rightarrow \mathrm{Observe1}(s,x) \vee \mathrm{Observe2}(s,y,x)$$

where Observe1(s,x) and Observe2(s,y,x) represent all possible means of achieving Observe(s,x).

$$\begin{aligned}
&\mathrm{Observe1}(s,x) \leftrightarrow \mathrm{Pointing}(s,x) \wedge \mathrm{TakeImg}(s,x) \wedge \\
&\quad \mathrm{start}(\mathrm{Pointing}(s,x)) \leq \mathrm{start}(\mathrm{TakeImg}(s,x)) \wedge \\
&\quad \mathrm{end}(\mathrm{Pointing}(s,x)) \geq \mathrm{end}(\mathrm{TakeImg}(s,x)) \wedge \\
&\quad \mathrm{start}(\mathrm{Observe1}(s,x)) = \mathrm{start}(\mathrm{TakeImg}(s,x)) \wedge \\
&\quad \mathrm{end}(\mathrm{Observe1}(s,x)) = \mathrm{end}(\mathrm{TakeImg}(s,x))
\end{aligned}$$

Observe1(s,x) happens if and only if the satellite is pointing at the target, the satellite takes an image, and the duration of the Observe1(s,x) action is the same as the TakeImg(s,x) action. This is true if and only if the satellite is already pointing at the target observation. Otherwise, a second constraint holds:

$$\begin{aligned}
&\mathrm{Observe2}(s,y,x) \leftrightarrow \mathrm{Pointing}(s,y) \wedge y \neq x \wedge \\
&\quad \mathrm{Slew}(s,y,x) \wedge \mathrm{TakeImg}(s,x) \wedge \\
&\quad \mathrm{end}(\mathrm{Pointing}(s,y)) = \mathrm{start}(\mathrm{Slew}(s,y,x)) \wedge \\
&\quad \mathrm{start}(\mathrm{Pointing}(s,x)) \leq \mathrm{start}(\mathrm{TakeImg}(s,x)) \wedge \\
&\quad \mathrm{end}(\mathrm{Pointing}(s,x)) \geq \mathrm{end}(\mathrm{TakeImg}(s,x)) \wedge \\
&\quad \mathrm{end}(\mathrm{Slew}(s,y,x)) < \mathrm{start}(\mathrm{TakeImg}(s,x)) \wedge \\
&\quad \mathrm{start}(\mathrm{Observe2}(s,y,x)) = \mathrm{start}(\mathrm{Slew}(s,y,x)) \wedge \\
&\quad \mathrm{end}(\mathrm{Observe2}(s,y,x)) = \mathrm{end}(\mathrm{TakeImg}(s,x))
\end{aligned}$$

where it requires that s be pointing at y, that y be different from x, that Slew(s,y,x) happen and that TakeImg(s,x) happen such that the Slew(s,y,x) happens before the TakeImg(s,x) and the duration of the Observe2(s,y,x) is the same as the duration of the Slew(s,y,x) and TakeImg(s,x) combined.

Finally, we include the mutual exclusion constraints that apply for any pair of observations that say that if two images are to be taken by the same satellite within a specific period of time there should be enough time to slew between them.

$$\begin{aligned}
&\mathrm{Observed}(s1,x1) \wedge \mathrm{Observed}(s2,x2) \rightarrow s1 \neq s2 \vee \\
&\quad (s1 = s2 \wedge x1 = x2) \vee \\
&\quad (s1 = s2 \wedge x1 \neq x2 \wedge \\
&\quad (\mathrm{end}(\mathrm{Observed}(s1,x1)) \leq \mathrm{start}(\mathrm{Observed}(s2,x2)) \\
&\quad \vee \mathrm{start}(\mathrm{Observed}(s1,x1)) \geq \mathrm{end}(\mathrm{Observed}(s2,x2)))
\end{aligned}$$

where if two observations happen then either they happen on different satellites or they are the same observation by the same satellite or the same satellite makes both observations but the observations don't overlap.

This constraint is possibly the most interesting and most complicated constraint in this example. Intuitively it is possible to derive the temporal constraint that allows for slewing between two consecutive tasks. This constraint is derived from the fact that slewing interferes with taking an image and vice-versa and the fact that two slewing actions interfere with each other. In the following section we show how. Note that once we have derived this constraint, we can do away with the variables for Slew(s,y,x) and Pointing(s,x) and the encoding is reduced to the Observe1(s,x) and Observe2(s,y,x) and the disjunctive constraint and this mutual exclusion constraint.

Compilation

For simplicity, we'll focus only on the predicates of interest: Pointing(s,x) and Vibrating(s) that are used to describe where a satellite is pointing and that a satellite vibrates during slew. A typical initial state will consist of (among other things) descriptions of where each particular satellite is pointing. A typical goal state will consist of descriptions of specific targets that should be observed. Given that the initial state contains Pointing(s,x) for a particular s,x it is possible to identify three cases: (1) the goal state includes Observed(s,x) for the same s,x; (2) the goal state includes Observed(s,y) where $y \neq x$; (3) there is no Observed(s,x) goal for that satellite. However, only (1) and (2) are of concern since they determine the cases that we need to consider when finding plans for individual goals. While this analysis may seem straight-forward, deriving this automatically may be difficult. However, we will show that it is possible to make use of a lifted plan graph in order to derive these constraints. As we will see a lifted plan graph is necessary for infering binding constraints that are used to deduce when all candidate plans have been obtained.

For simplicity of exposition, we make the following assumptions: (1) each satellite has a single instrument that can be used to observe a target, thus allowing us to abstract out the instrument; (2) opportunities for taking an image can be computed and compiled by limiting the possible times at which a satellite may be pointing to an image; (3) an initial description of the problem and initial state is given in PDDL 2.1. We further assume that resources can be expressed in the language natively (see Bedrax-Weiss, McGann, & Ramakrishnan 2003 for examples) and dealt with separately. We also assume that the availability of targets is modeled as exogenous events (see Edelkamp & Hoffmann 2004 for examples). Exogenous events are events that the planner has no control over and are similar to availability dates in scheduling. Following these assumptions, a simplified PDDL 2.1 description for the EOS domain would include:

```
(:durative-action Slew
 :parameters ( ?s - satellite
               ?x - observation
               ?y - observation)
 :duration (= ?duration
              (/ (DistanceBetween ?x ?y)
                 (SlewRate ?s)))
 :condition ( and (at start (Pointing ?s ?x)))
 :effect (and (at start (Vibrating ?s))
              (at end (not (Vibrating ?s)))
              (at start (not (Pointing ?s ?x)))
              (at end (Pointing ?s ?y))
         )
)

(:durative-action TakeImg
 :parameters (?s - satellite
              ?x - observation
              ?w - window)
 :duration (= ?duration (DurationOf ?o))
 :condition (and (at start (TargetAvailable ?w))
                 (over all (TargetAvailable ?w))
                 (at start (Pointing ?s ?x))
                 (over all (Pointing ?s ?x))
                 (at end (Pointing ?s ?x))
                 (at start (not (Vibrating ?s)))
                 (over all (not (Vibrating ?s)))
                 (at end (not (Vibrating ?s)))
            )
 :effect (and (at end (Observed ?x)))
)
```

We begin by lifting the initial and goal conditions, keeping the corresponding set of variable bindings for each condition. For example given Pointing(sat1,tgt1) we abstract out the constants and replace them with variables and the corresponding binding constraint: Pointing(s,x) and $\{s = sat1, x = tgt1\}$. We do the same for goals: Observed(tgt1) turns into Observed(x) and $\{x = tgt1\}$.

A lifted plan graph is constructed for each observation sub-goal by adding the lifted initial conditions to the graph. The plan graph will contain lifted instances of actions and propositions. An action will only be added if all conditions are satisfied. We say that a condition is satisfied if a unification constraint exists between the condition and a condition already in the graph. We add all lifted actions such that:

1. Start conditions can be satisfied by lifted conditions in the graph
2. Overall conditions can be satisfied either by lifted conditions in the graph or by start effects of the lifted action.
3. End conditions can be satisfied by lifted conditions in the graph or effects of the lifted action.

For each action, we derive the set of unification constraints that would make that action applicable. For instance, if the initial condition is Pointing(s1,x1) and an action TakeImg has a start condition that says Pointing(s2,x2), the unification constraints would be $\{s1 = s2, x1 = x2\}$. Furthermore, if TakeImg has an effect Observe(x2) and there's a goal condition Observe(t3), we derive the unification constraint $\{x2 = x3\}$. Together, these unification constraints would imply that $\{x1 = x3\}$ which allows us to derive that if there is an initial condition that states that a satellite is pointing to a target and there is a goal condition that states that the target is to be observed, then the TakeImg action satisfies that goal. Furthermore, we derive that if $\{x1 \neq x3\}$ then TakeImg alone does not satisfy the goal and more work needs to be done to find a plan. In this case, we continue expanding the plan graph ignoring action durations and mutex propagations. Durations will come into play when we analyze the interactions between the different plans for achieving the lifted goals.

Since in the case of observations there are only two possible plans for each sub-goal depending on whether the satellite is already pointing at the target or whether it needs to slew, we stop generating plan graphs when we have exhausted both cases. In other domains there may be more conditions that will arise from binding constraint analysis of the initial and goal states. For this domain, though we finalize expansion with two "candidate plans":

If x1 = x3 then TakeImg(s,x1)
Else Slew(s,x1,x3) then TakeImg(s,x3).

Once we have all alternative "candidate plans" for each sub-goal, we form the disjunctive constraint that shows all of the possible ways to achieve each individual goal. Next, we derive lifted static mutual exclusion constraints as follows.

For each possible way of overlapping actions, we analyze all possible overlaps allowed by the PDDL 2.1 semantics. Two actions may overlap if there are no contradictions in the static analysis of the states. The static analysis is done in a way similar to (Garrido, Fox, & Long 2002; Garrido & Onandía 2003). For instance, if two actions are allowed to start together, then the start conditions of both actions are simultaneously satisfied, the start effects of both actions do not interfere, and the start effects of one action do not interfere with the overall conditions of the other. We need to find out whether given the two candidate plans found above, can Slew and TakeImg overlap? Can TakeImg and

TakeImg overlap? We illustrate how the technique works by analyzing whether Slew and TakeImg can start at the same time.

In order to determine whether Slew and TakeImg can start at the same time, we must determine that:

1. The start conditions of both actions are simultaneously satisfied.
2. The start effects of both actions do not interfere.
3. The start effects of one action do not interfere with the overall conditions of the other.

Interference in the lifted case is determined by the set of binding constraints that would cause an inconsistency. This is easy enough to determine because these binding constraints can be computed by intersecting variable domains. There are two ways an inconsistency could be generated: (1) when computing the binding constraints, the intersection of the domains is empty; (2) one of the conditions is the negation of the other. In this latter case the binding constraints determine the cases where there is interference. Let us examine the start and overall conditions and start effects of both actions:

```
Slew(?s ?x1 ?x2)
  :condition ( and (at start (Pointing ?s ?x1)))
  :effect (and (at start (Vibrating ?s)))

TakeImg(?s ?x)
  :condition (and (at start (TargetAvailable ?w))
                  (over all (TargetAvailable ?w))
                  (at start (Pointing ?s ?x))
                  (over all (Pointing ?s ?x))
                  (at start (not (Vibrating ?s)))
                  (over all (not (Vibrating ?s)))
             )
```

TakeImg doesn't have any start effects so we only need to worry about the start effect of the Slew action. Furthermore, only the Vibrating start and overall condition of TakeImg can produce interference since they appear negated in Slew. Thus, of the three possibilities for interference namely, start conditions interfering with each other, or start effects interfering with each other or start effects interfering with start conditions, only the third leads to interference. The start effect of Slew interferes in one way with the overall conditions of TakeImg through the Vibrating predicate — Slew asserts (Vibrating ?s) at the start and TakeImg requires the negation of it whenever the satellites are the same. We have just learned one condition for interference: the satellites are the same. If this condition is satisfied, Slew and TakeImg cannot start at the same time. So the initial constraint is:

$$\begin{aligned}&\mathrm{Slew}(s1,x1,x2) \wedge \mathrm{TakeImg}(s2,x3) \rightarrow s1 \neq s2 \vee\\&(s1 = s2 \wedge\\&(\mathrm{start}(\mathrm{Slew}(s1,x1,x2)) < \mathrm{start}(\mathrm{TakeImg}(s2,x3)) \vee\\&\mathrm{start}(\mathrm{Slew}(s1,x1,x2)) > \mathrm{start}(\mathrm{TakeImg}(s2,x3))))\end{aligned}$$

Now we know that they can't start at the same time, but can TakeImg start during Slew? One of the things we need to determine is whether the start effects of Slew interfere with the conditions of TakeImg. Slew has two start effects: (Vibrating ?s) and (not (Pointing ?s ?x1)). TakeImg has the following conditions: (not (Vibrating ?s)) and (Pointing ?s ?x), both of which interfere under the following conditions: the satellites are the same and $\{x1 = x\}$ (according to Pointing(...)). However, this constraint is subsumed by the constraint imposed by Vibrating(...) that says that the satellites have to be different. So the following constraint we derive is:

$$\begin{aligned}&\mathrm{Slew}(s1,x1,x2) \wedge \mathrm{TakeImg}(s2,x3) \rightarrow (s1 \neq s2) \vee\\&(s1 = s2 \wedge x1 = x3 \wedge\\&(\mathrm{end}(\mathrm{Slew}(s1,x1,x2)) < \mathrm{start}(\mathrm{TakeImg}(s2,x3)) \wedge\\&(\mathrm{start}(\mathrm{Slew}(s1,x1,x2)) > \mathrm{end}(\mathrm{TakeImg}(s2,x3))))\end{aligned}$$

After analyzing all of the possible ways TakeImg and Slew can overlap, we can derive conditions that when combined with all other conditions that are derived, imply that the sequence Slew,TakeImg cannot overlap the sequence of TakeImg unless the satellites are different. The combination of constraints is done by grouping implications and substituting terms for equivalent terms to yield a more compact representation:

$$\begin{aligned}&\mathrm{Observed}(s1,x1) \wedge \mathrm{Observed}(s2,x2) \rightarrow s1 \neq s2 \vee\\&(s1 = s2 \wedge x1 = x2) \vee\\&(s1 = s2 \wedge x1 \neq x2 \wedge\\&(\mathrm{end}(\mathrm{Observed}(s1,x1)) < \mathrm{start}(\mathrm{Observed}(s2,x2))\\&\vee \mathrm{start}(\mathrm{Observed}(s1,x1)) > \mathrm{end}(\mathrm{Observed}(s2,x2)))\end{aligned}$$

The same analysis can be used to determine whether the windows of opportunity interfere with observations. That is because windows of opportunity are modeled as exogenous events. In their simplest way, an exogenous event can be translated to an action with durations where the duration is the time at which the event happens (a precondition and effect is created to ensure that the event cannot happen more than once, if that is the desired result). Interference analysis for these actions allows us to derive the constraint that says that observations must happen within the window of opportunity.

Note that Slew and TakeImg are no longer involved in any of the constraints. Slewing is only relevant in that it influences the time at which tasks can be performed. Therefore no explicit representation of these actions is needed. Note that the encoding deals with temporal flexibility just like scheduling deals with temporal flexibility in terms of start time and end time for each task (Crawford 1996). If we ignore resources, it is possible to see that the temporal constraint is sufficient in order to find a solution to the problem. With respect to the resource consumption we postulate that since the problem is being compiled into a scheduling problem, resources can be handled directly in the scheduling formulation. This encoding is based on the fact that determining the sequence and assignment of the different image requests depends on an explicit representation of mutually exclusive sets of decisions. For example, given an image request, there is a set of mutually exclusive options for assigning the request to the satellite.

Advantages of Automated Compilation

There are several advantages to using automated compilation. No known encoding methods can deal with complex constraints, resources, and actions with durations. Since we will be encoding the problem as a CSP that resembles a scheduling problem, our encoding will deal naturally with these features. In recent years, SAT planners have not been competitive with other approaches to planning. In the 2000 competition, SAT planners ran out of memory because their encodings were too large. In the 2002 competition, none of the planners were SAT based (see results on planning competitions published online at Long & Fox 2003 and http://www.icaps-conference.org/). Furthermore, these encodings couldn't deal well with metric time. Most encodings had to encode explicit time choices as variables and because they were being encoded as SAT would have one variable for each action choice per time point. Even if these encodings were sent to a constraint satisfaction engine that expanded the domains lazily the encodings were not able to handle durations, timed events, or resources. Because our encoding looks like a scheduling problem, metric constraints and resources can be dealt with naturally.

Our encoding more accurately represents the actual choices that are made in the domain and therefore is more able to successfully guide the search for plans. Our encoding looks quite different from the planning encoding of the same problem. There are no pre-conditions, post-conditions, sub-goals, fluents (other than resources), etc. Rather, the choice of which observation window to put each image request into looks much more like a scheduling problem. Furthermore, arc-consistency, and other CSP and scheduling propagation techniques, can be used to quickly compute the consequences of a set of variable bindings. Because of the strength of these propagation techniques, because of the reduction in the number of variables, and because the key optimization variables are "obvious" to the search algorithm, this CSP encoding we postulate that it allows much more efficient search and optimization for EOS planning. In one instance, a planning system took roughly an hour to solve an EOS problem with 400 requests and an average domain size (start times for the requests) of about 10. Tests done on an encoding similar to the one we propose in this work show a reduction in run time to about .2 seconds.

Some additional advantages of this versus other approaches are:

1. Our encoding makes the scheduling decisions explicit. It provides more direct guidance to the constraint engine because we have chosen to abstract and learn a set of constraints that define the interactions among the different means of achieving a sub-goal and then use these constraints rather than the planning structure to guide the constraint engine.
2. The lifted nature of the approach makes the encoding more compact. Furthermore, it allows us to derive the binding constraints that ultimately lead to the mutual exclusion constraints between sub-plans.
3. Unnecessary variables describing the causality of the planning problem also disappear in our encoding making it more compact.
4. In other approaches resources would have to be captured by creating n-way mutual exclusions for resources of capacity n (i.e., not more than n of i1=t1, i2=t2, i3=t3,...), obscuring the description and creating unnecessary variables.

We postulate that this is not an isolated event — that there is a broad class of problems that can be compiled into static CSPs using non-trivial but still automated mapping.

Conditions for Automatic Compilation

For a planning problem to be compiled into a static CSP using the proposed method it must meet the following tests:

1. It must have many weakly interacting goals or sub-goals. For example, in EOS scheduling the image requests from the scientists only interact in their use of shared resources (cameras, SSR, etc.). In contrast, a set of blocks to be stacked ABCDE represents a set of goals that interact with each other in complex and subtle ways (e.g., putting D on C prevents C from being moved).
2. Each goal can be achieved in a reasonably small number of ways. For example, in EOS the number of observation windows for each image request is small. In contrast, if a goal can be achieved in several ways, each of which consists of sub-goals that can each be achieved in several ways, then the total number of possible unique ways to achieve a goal can become quite large. The proposed encoding will create a variable for each goal and a value for each qualitatively unique way to achieve the goal (in isolation). If goals can be achieved in many different ways then the number of values can become prohibitive.
3. Structure sharing between the plans for different goals must be minimal. This is an important variant of condition 1. For example, in some planning problems it may be the case that achieving goal A using plan fragment p1 makes it very easy to achieve goal B (since much of p1 can be shared by a plan fragment for B). This kind of positive interaction is difficult to capture in a general form in the proposed framework (though some special cases, such as sequence dependent setup, can be handled in the same way as in scheduling problems).
4. Goals are non-cyclic. That is, the sub-goal tree for a goal g must not contain g itself. Note that while such cycles are rare in problems like EOS scheduling they are pervasive in more traditional planning problems (e.g., blocks world). Some kinds of cycles can be broken by careful manipulation of the planning encoding but others are fundamental to the planning domain.

If the goal network is non-cyclic we believe that it is always possible to push conditions to the sub-plan level rather than retaining the details of how the plan is achieved. This does not change the problem but it simplifies thinking about it. In the satellite example, we saw that it is possible to push conditions for slews, take images, etc. to the top level Observe action. Furthermore, we note that the SSR requirement imposes a "global" constraint on the way observations are carried out. However, this global constraint can be captured as a resource (whose capacity is refilled on downlink). The lack of strong interaction makes this a feasible approach.

Discussion

There are many challenges in this proposed work. It may be difficult to prove that the static CSP encapsulates all of the constraints in the problem. If compilation is not possible for certain domains, then automatically identifying a subset of the problem with the required properties will also be a challenge. Furthermore, even if it was possible to automatically identify the sub-problem, will solving the sub-problem separately help the planner's overall performance? A similar question arises in the case where it is possible to compile an approximation to the original problem but not the problem itself. In what follows, we provide some answers to these questions.

One difficulty lies in formally proving that all qualitatively distinct sub-plans that could possibly achieve a given sub-goal have been enumerated. (Though note that even in the absence of such a proof the solution to the CSP could still serve as a very strong heuristic[1] in solving the full planning problem). For example, for the EOS problem domain, the obvious sub-plans are those that call for a single slew from each target to the next target. There are, however, an infinite number of distinct sub-plans that call numerous small slews. To justify ignoring these variants we have to prove that you'll never be required to do anything but the shortest path slew. A satellite might be required to break up the shortest path slew if there was an obstruction along the path. For example there may be obstructions such as a physical obstacle on the satellite itself or an external object that would damage the instrument if pointed at it, e.g. the Sun. Both of these are flight safety rules that could interfere with the shortest path slewing. However, both of them can be compiled because there is a limited set of circumstances under which the obstruction occurs and those circumstances can be determined at compile-time (we know where the Sun is).

The general challenge is in proving that all interesting qualitatively distinct sub-plans have been included. Note, however, that a human creating a CSP or scheduling encoding is implicitly completing such proofs in their head. Also note that failing to prove that we have enumerated all possible sub-plans impacts only the optimality of our approach with respect to plan length (or slew time in our case) - not its correctness. That is, all solutions to CSP(P) will be valid solutions of P, but there is a possibility that there are additional sub-plans for some sub-goals in P that would have yielded a more optimal plan.

One final question to consider is the tradeoff between compiling planning problems into CSPs vs. directly encoding the problem as a CSP. Clearly there is experience in the CSP and scheduling community on directly encoding problems of realistic size and complexity. Further, the direct encoding has the advantage that the modeler can more carefully control and optimize the choice of variables and the representation of constraints. Also, the direct method obviously makes the construction of an automatic compilation system unnecessary. However, the use of a planning encoding followed by automated compilation has several advantages: (1) In many cases the planning encoding is more natural, and more accessible and intuitive for domain experts. In the case of rover ground operations, for example, the Ames team building the planner for the MER rovers has been able to largely automate the translation from the "data dictionary" (which is actually expressed in a sub-goal format) produced by the flight team to a planning encoding. This would probably not be possible for a hand-built CSP encoding. This makes the planning encoding easier to both validate and to maintain. (2) An automated translation will identify any hidden goal interactions that might be missed in a hand-built CSP encoding. In the hand encoding the human modeler is implicitly computing all interactions between the tasks and accounting for them by creating constraints or adding resources. In the automated method the human modeler specifies the sub-goal structure and the preconditions and effects of each action. The compilation procedure then identifies all possible interactions. (3) The compilation mechanism provides a relatively low cost way to generate heuristics that may help in solving the full planning problem (by first solving the CSP and then using that solution as a guide in solving the full problem). Finally, (4) the direct encoding is brittle in the sense that if a constraint changes slightly in the original problem it may force substantial changes in the compiled problem. In particular, a relatively modest change to the problem statement may have complex and easily-missed effects on the task interactions. For example, if one adds the flight rule that slewing one instrument platform will create vibrations in all other cameras on the satellite, then a host of previously unrelated tasks will become mutually incompatible. Note finally that the human encoder must add constraints that capture each possible pairwise interaction between sub-goals and in some cases there will be three-way or higher interactions. The difficulty of doing this by hand will obviously increase quickly with domain complexity.

[1]The CSP solution will be mostly right so it could be queried by a search engine in order to determine the next decision to make.

Related Work

One approach in HTN planning that we think may be applicable here is Barrett's (Barrett 1997) work which provides a method to guarantee plan decompositions are frugal (e.g. every action contributes positively; negative contribution is minimized). This work may prove relevant when the number of ways of achieving individual goals grows large by helping narrow down the set of sub-plans we will consider in the encoding. Plan decompositions in HTN are reminiscent of the "flattening" technique since they also explicitly encode the choice of how to achieve an individual goal (Mali & Kambhampati 1998).

Another relevant body of work is the Fox and Long (1998) approach of building finite state machines describing the evolution of conditions in the domain in order to recognize patterns of evolution that arise in different domains and reformulate the portions of the domain that fit the pattern into a problem that can be solved using domain specific techniques. Their approach seems to break down when the finite state machine generated for a condition differs slightly from a recognized pattern. For instance, when moving from point A to point B involves stopping along the way and the available state machine represents a direct movement from point A to point B. Even though their approach is not generally applicable, we may be able to leverage some of the lessons learned and apply them to our research in order to perform domain analysis to determine if the problem exhibits the characteristics required to automatically compile it to a static CSP.

Becker and Smith (2000) describe an approach whereby they generate plans for each individual sub-goal and then concatenate them to form plans for the "mission". Although they do not generate all possible plans for each sub-goal they do impose the constraint that there must be enough resources to satisfy both plans whenever they combine any two plans. We are not manually imposing this constraint, we are deriving it via mutual exclusion reasoning.

Automatically generating problem transformations is another challenge. There have been other approaches to automatically generating problem description transformations in order to more efficiently solve the original problem.

Van Baalen (1992) uses a set of sentence schemas defining common concepts from set theory (symmetry, reflexivity, transitivity, etc.) to reformulate a problem given in predicate calculus with the objective of maximizing the compression of the reformulation. This approach, however, is only applicable to problem instances of the same class, that is, where the set of actual domains change but not the domain rules. This approach is related to our approach because it provides some means to learn in the lifted domain. We believe working in the lifted domain will allow us to explore the possibility of extending our approach to cases where there are many scheduling choices.

Acknowledgements

Thanks to Al Globus for discussions on EOS. This research was supported by NASA Ames Research Center and the NASA Intelligent Systems program.

References

Barrett, A. C. 1997. *Hierarchical Task Network Planning Using Actions With Universally-Quantified Conditional Effects*. Ph.D. Dissertation, University of Washington.

Becker, M. A., and Smith, S. F. 2000. Mixed-initiative resource management: The amc barrel allocator. In *AIPS*.

Bedrax-Weiss, T.; Jónsson, A.; and Ginsberg, M. 1996. Unsolved problems in planning as constraint satisfaction. http://www.cirl.uoregon.edu/tania/html_files/pubs.html.

Bedrax-Weiss, T.; McGann, C.; and Ramakrishnan, S. 2003. Formalizing resources for planning. In *ICAPS Workshop on PDDL*.

Crawford, J. M. 1996. An approach to resource constrained project scheduling. In *Workshop on AI and Manufacturing Research Planning*.

Do, M. B., and Kambhampati, S. 2000. Solving planning-graph by compiling it into CSP. In *AIPS*.

Dungan, J.; Frank, J.; Jónsson, A.; Morris, R.; and Smith, D. 2002. Advances in planning and scheduling of remote sensing instruments for fleets of earth orbiting satellites. In *ESTC*.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2:the language for the classical part of the 4th international planning competition. http://ipc.icaps-conference.org/.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in tim. *Journal of AI Research* 9.

Fox, M., and Long, D. 2003. Pddl2.1: An extension of pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20.

Garrido, P., and Onandía, E. 2003. On the application of least-commitment and heuristic search on temporal planning. In *IJCAI*.

Garrido, P.; Fox, M.; and Long, D. 2002. Temporal planning with PDDL2.1. In *ECAI*.

Globus, A.; Crawford, J.; Lohn, J.; and Morris, R. 2002. Scheduling earth observing fleets using evolutionary algorithms: Problem description and approach. In *3rd NASA Planning and Scheduling Workshop*.

Globus, A.; Crawford, J.; Lohn, J.; and Pryor, A. 2004. A comparison of techniques for scheduling earth observing satellites. In *IAAI*.

Haas, A. 1987. The case for domain-specific frame axioms. In *The Frame Problem in Artificial Intelligence Workshop*.

Kautz, H., and Selman, B. 1998. Blackbox: A new approach to the application of theorem proving to problem solving. In *AIPS*.

Kautz, H.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Principles of Knowledge Representation and Reasoning*.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research* 20.

Mali, A. D., and Kambhampati, S. 1998. Encoding HTN planning in propositional logic. In *AIPS*.

McAllester, D., and Rosenblitt, D. 1991. Sistematic nonlinear planning. In *AAAI*.

Morris, R.; Dungan, J.; Frank, J.; Khatib, L.; and Smith, D. 2003. An integrated approach to earth science observation scheduling. In *ESTC*.

Smith, D. E. 2003. The case for durative actions: A commentary on PDDL2.1. *JAIR* 20.

van Baalen, J. 1992. Automated design of specialized representations. *Artificial Intelligence* 54(1–2).